Correction Creating and using Arrays - 0

B - Line 3 causes compilation error

Correction Creating and using Arrays - 1

C - Only three options

Let's check all the statements one by one:

1. short arr [] = new short[2]; => ✗

You can declare one-dimensional array by using either "short arr []" or "short [] arr". 'arr' refers to a short array object of 2 elements. arr[2] will throw ArrayIndexOutOfBoundsException at runtime.

2. byte [] arr = new byte[10]; => ✓

'arr' refers to a byte array object of 10 elements, where 0 is assigned to each array element. But later on element at 1st and 2nd indexes have been re-initialized. Line n1 successfully prints [5, 10] on to the console.

3. short [] arr; arr = new short[3]; => ✓

You can create an array object in the same statement or next statement. 'arr' refers to a short array object of 3 elements, where 0 is assigned to each array element. Later on element at 1st and 2nd indexes have been re-initialized. Line n1 successfully prints [5, 10] on to the console.

4. short [2] arr; ✗

Array size cannot be specified at the time of declaration, so short [2] arr; causes compilation error.

5. short [3] arr; ✗

Array size cannot be specified at the time of declaration, so short [3] arr; causes compilation error.

6. int [] arr = new int[]{100, 100}; => ✗

'arr' refers to an int array object of size 2 and both array elements have value 100. arr[2] will throw ArrayIndexOutOfBoundsException at runtime.

7. int [] arr = new int[]{0, 0, 0, 0}; => ✓

'arr' refers to an int array object of size 4 and all array elements have value 0. Later on element at 1st and 2nd indexes have been re-initialized. Line n1 successfully prints [5, 10] on to the console.

8. short [] arr = {}; => ✗

'arr' refers to a short array object of 0 size. so arr[1] will throw ArrayIndexOutOfBoundsException at runtime.

9. short [] arr = new short[2]{5, 10}; => ✗

Array's size can't be specified, if you use {} to assign values to array elements.

Correction Creating and using Arrays - 2

A - int arr4[][] = new int[][8];

Correction Creating and using Arrays - 3

C - All four options

From the given array, if you print the elements at 1st, 3rd and 5th indexes, then you will get expected output.

Also note that, for values of n = 0, 2, 4, 6; Line n1 would not be executed, which means even if the value of n is 6, above code will not throw ArrayIndexOutOfBoundsException.

For 1st option [int n = 0; n < arr.length; n += 1], values of n used: 0, 1, 2, 3, 4, 5 and because of continue; statement, Line n1 will not execute for 0, 2 & 4 and it will execute only for 1, 3 & 5 and therefore NET will be printed.

For 2nd option [int n = 0; n <= arr.length; n += 1], values of n used: 0, 1, 2, 3, 4, 5, 6 and because of continue; statement, Line n1 will not execute for 0, 2, 4 & 6 and it will execute only for 1, 3 & 5 and therefore NET will be printed.

For 3rd option [int n = 1; n < arr.length; n += 2], values of n used: 1, 3, 5 and therefore NET will be printed.

For 4th option [int n = 1; n <= arr.length; n += 2], values of n used: 1, 3, 5 and therefore NET will be printed.

Correction Creating and using Arrays - 4

D - int [] arr3 = new int[3]{10, 20, 30};

You can’t specify size at the time of initializing with data, hence new int[3] {10, 20, 30}; gives compilation error.

Correction Creating and using Arrays - 5

B - Some String containing @ symbol

Variable arr refers to an array object of String of 7 elements.

Variable arr contains the memory address of String array object.

arr is of reference type, hence it prints some String Containing @ symbol.

Correction Creating and using Arrays - 6

C - 0

We have not passed any command-line arguments, hence args refers to an array object of Size 0.

args.length prints 0. args is not null and hence no NullPointerException.

Also we are not accessing array element so no question of ArrayIndexOutOfBoundsException as well.

Correction Creating and using Arrays - 7

H - 642

case values must evaluate to the same type / compatible type as the switch expression can use.

switch expression can accept following:

char or Character,

byte or Byte,

short or Short,

int or Integer,

An enum only from Java 6,

A String expression only from Java 7.

In this case, switch expression [switch (arr[i][j])] is of String type.

Please note that break; statement at Line n4 takes the control to Line n6 (outside switch-case block) and not out of the inner for loop, where as, continue; statement at Line n5 takes the control to the update expression (j++) of Line n2.

arr.length is 3, so outer loop executes 3 times.

1st iteration of outer loop, i=0.

-1st iteration of inner loop, i=0, j=0 and arr[0].length = 3. 0 < 3 evaluates to true.

arr[0][0] = “7”, Line n5 is executed, and it takes the control to j++ (j = 1). 1 < 3 evaluates to true.

-2nd iteration of inner loop

arr[0][1] = “6”, Line n4 is executed, and it takes the control to Line n6. 6 is printed on to the console. Control goes to j++ (j = 2). 2 < 3 evaluates to true.

-3rd iteration of inner loop

arr[0][2] = “5”, Line n5 is executed, and it takes the control to j++ (j = 3). As 3 < 3 evaluates to false, control exits inner loop and goes to i++.

You must have noticed that 1st iteration of outer loop prints the even number of 1st array { “7”, “6”, “5” }

Similarly, 2nd iteration of outer loop prints the even number of 2nd array { “4”, “3” }, which is 4

and 3rd iteration of outer loop prints the even number of 3rd array { “2”, “1” }, which is 2.

Therefore, the output is: 642.

Correction Creating and using Arrays - 8

E - Only Five blocks

There are 2 ways to change the value of count variable of Counter class:

1. As access modifier of count variable is public, hence it can easily be accessed from other classes using the instance of Counter class, such as `new Counter().count` or `obj.count` (where obj is reference variable of Counter type, referring to Counter variable's instance)
2. By invoking the increase(int) method of Counter class.

Now let's check all the blocks one by one:

1.

```
for(Counter ctr : arr) {
    ctr.count = 100;
}
```

✓ It will assign 100 to count variables of three instances of Counter class.

2.

```
for (Counter ctr : arr) {
    int x = ctr.getCount();
    x = 100;
}
```

✗ x is local variable and is copy of ctr.count. Hence, assigning 100 to x will not affect the value of ctr.count.

3.

```
for (Counter ctr : arr) {
    ctr.getCount() = 100;
}
```

✗ ctr.getCount() returns int value and not a variable, hence cannot be used on left side of assignment operator. It causes compilation error.

4.

```
for(Counter ctr : arr) {
    ctr.increase(100 - ctr.count);
}
```

✓ You must have noticed that value of count variable of 3 array elements are: -1000, 539, 0. How will you change all 3 values to 100 using same expression? It is by adding 100 and subtracting current value. For example,

-1000 + 100 -(-1000) = 100

or

539 + 100 - 539 = 100

or

0 + 100 - 0 = 100

And same this is done by executing `ctr.increase(100 – ctr.count);` statement.

5.

```
for (Counter ctr : arr) {
    ctr.increase(100 – ctr.getCount());
}
```

✓ Same as block no. 4. Only difference is ctr.getCount() is used instead of ctr.count.

6.

```
for(Counter ctr : arr) {
    ctr.increase(–ctr.getCount() + 100);
}
```

✓ Same as block no. 5.

7.

```
for(Counter ctr : arr) {
    ctr.increase(–ctr.count + 100);
}
```

✓ Same as block no. 4.

Hence, out of given 7 blocks, 5 will give you expected output.

Correction Creating and using Arrays - 9

B - 4

Initially arr1 refers to an int array object of 3 elements.

And arr2 refers to an int array object of 2 elements [char type is compatible with int type]

When the statement `arr1 = arr2;` executes, variable arr1 copies the content of arr2, which is the address of array object containing 2 elements. Hence, arr1 also starts referring to same array object. arr1.length = 2 and arr2.length = 2.

Therefore, output is: 4

Correction Creating and using Arrays - 10

D - ArrayIndexOutOfBoundsException

arr refers to an array object of size 0. That means arr stores some memory address.

So we will not get NullPointerException in this case. But index 0 is not available for an array object of size 0 and thus ArrayIndexOutOfBoundsException is thrown at runtime.

Correction Creating and using Arrays - 11

D - NullPointerException is thrown at runtime

All the elements of array are initialized to respective zeros (in case of primitive type) or null (in case of reference type).

So, arr[0] refers to null.

Method ‘isEmpty()’ is invoked on null reference and hence NullPointerException is thrown at runtime.

Correction Creating and using Arrays - 12

B - Hello

Variable msg is referring to String object “Hello”.

There is only one element in boolean array object and it is initialized to default value of boolean, which is false.

flag[0] is false, if-check fails and control doesn’t enter if block.

System.out.println(msg) prints original value of msg, which is “Hello”.

Correction Creating and using Arrays - 13

B -

```
int n = 0; n < arr.length; n += 2
```

D -

```
int n = 3; n < arr.length; n++
```

E -

```
int n = 1; n < arr.length - 1; n++
```

Logic in for loop is adding array elements. You need to find out which array elements when added will result in 9. Possible options are: {1+3+5, 2+3+4, 4+5}.

Based on these 3 combinations you can select 3 correct options.

Correction Creating and using Arrays - 14

A - Compilation error

Initially arr1 refers to an int array object of 3 elements: 1, 2, 3

And arr2 refers to an char array object of 2 elements: ‘A’, ‘B’.

Statement arr1 = arr2; gives compilation error as char [] is not compatible with int [] even though char is compatible with int.

Correction Creating and using Arrays - 15

C - B E C D

arr[0] -> “A” and arr[1] -> “B”.

arr[0] = arr[1]; => arr[0] -> “B” and arr[1] -> “B”.

arr[1] = “E”; => arr[0] -> “B” and arr[1] -> “E”.

Hence output is: B E C D

Correction Creating and using Arrays - 16

D -

```
BBB
EEE
HHH
```

NOTE: System.out.print statement is printing arr[i][1],

which means it prints 2nd array element of a particular row, for each iteration of inner loop.

That is why output is:

```
BBB
EEE
HHH
```

To get all the array elements printed correctly, use arr[i][j] in System.out.print statement.

Correction Creating and using Arrays - 17

A - Compilation error

Initially arr1 refers to an int array object of 3 elements: 1, 2, 3

And arr2 refers to an char array object of 2 elements: ‘A’, ‘B’.

Statement arr1 = arr2; gives compilation error as char [] is not compatible with int [] even though char is compatible with int.

Correction Creating and using Arrays - 18

C - B E C D

arr[0] -> “A” and arr[1] -> “B”.

arr[0] = arr[1]; => arr[0] -> “B” and arr[1] -> “B”.

arr[1] = “E”; => arr[0] -> “B” and arr[1] -> “E”.

Hence output is: B E C D

Correction Creating and using Arrays - 19

D -

```
BBB
EEE
HHH
```

NOTE: System.out.print statement is printing arr[i][1],

which means it prints 2nd array element of a particular row, for each iteration of inner loop.

That is why output is:

```
BBB
EEE
HHH
```

To get all the array elements printed correctly, use arr[i][j] in System.out.print statement.

Correction Creating and using Arrays - 20

B - It prints some text containing @ symbol

Variable ‘arr’ refers to an array object of String of 7 elements and it contains the memory address of String array object.

‘arr’ is of reference type, therefore when `System.out.println(arr);` is executed, toString() method defined in Object class is invoked, which returns @. That is why some text containing @ symbol is printed on to the console.

Correction Creating and using Arrays - 21

A - All 7 pairs

Given question expects you to solve the compilation error and not care about runtime error. For array indexes, any int values can be used, hence all the 7 pairs are allowed in this case.

If question were expecting to compile and execute the program successfully, then any combination greater than the max indexes values would have worked. For example, in the given code, as max 1st dimension value = 6 and max 2nd dimension value = 6, so any int value > 6 can be used for x and any int value > 6 can be used for y.

Out of the given seven options, only two options (x = 7, y = 7) and (x = 8, y = 8) would have worked.

Correction Creating and using Arrays - 22

B - LOVE

Line n1 creates a String [] object of 4 elements and arr refers to this array object. arr[0] = “L”, arr[1] = “I”, arr[2] = “V” and arr[3] = “E”.

i = -2.

Boolean expression of Line n2: i++ == -1

=> (i++) == -1 //As Post-increment operator ++ has higher precedence over ==

=> -2 == -1 //i = -1, value of i is used in the expression and then incremented.

=> false and hence Line n3 is not executed.

But there is no issue with Line n3 and it compiles successfully.

Boolean expression of Line n4 is evaluated next:

–i == -2 //i = -1

=> (–i) == -2 //As Pre-decrement operator – has higher precedence over ==

=> -2 == -2 //i = -2, value of i is decremented first and then used in the expression.

=> true and hence Line n5 is executed next.

Line n5:

arr[-++i] = “O”; //i = -2

=> arr[-(++i)] = “O”; //Unary minus ‘-’ and pre-increment ‘++’ operators have same precedence

=> arr[-(-1)] = “O”; //i = -1, value of i is incremented first and then used in the expression.

=> arr[1] = “O”; //2nd array element is changed to “O”.

Hence after Line n5, arr refers to {“L”, “O”, “V”, “E”}

Given loop prints LOVE on to the console.

Correction Creating and using Arrays - 23

D - It throws an exception at runtime

Line n1:

String [][] arr = new String[–var][var++]; //var = 3

Access array element operator [] is left to right associative.

=> String [][] arr = new String[2][var++]; //var = 2, var is decremented first and then used in the expression.

=> String [][] arr = new String[2][2]; //var = 3, value of var is used first and then it is incremented by 1

Hence, arr refers to 2-dimensional String array object {{null, null}, {null, null}}.

At Line n2, arr[1][1] = “X”; assigns “X” to element at index [1][1], therefore arr –> {{null, null}, {null, “X”}}

At Line n3, arr[1][2] = “Y”; causes ArrayIndexOutOfBoundsException as 2nd index 2 is out of range.

As Line n3 throws Exception at runtime, hence for loop will not be executed.

Correction Creating and using Arrays - 24

F - Line n2 causes compilation error

arr1 is of String[] type, where as arr2 and arr3 are of String type. As all three arr1, arr2 and arr3 are of reference type, hence null can be assigned to all these variables. Line n1 compiles successfully.

Statement at Line n2: arr2 = arr3 = arr1;

=> arr2 = (arr3 = arr1); //assignment operator is right to left associative.

arr3 is of String type and arr1 is of String [] type, hence (arr3 = arr1) causes compilation error.

Though you had to select one correct option, hence no need to look further but I am providing explanation for Line n3 as well.

log(String…) method can be called using a String [] or a String instance or mutliple String instances: log(new String[] {“A”, “B”}); log(“A”); log(“A”, “B”);

As arr2 is of String type, hence `log(arr2);` (Line n3) compiles successfully.

Creating and using Arrays - 25

C - 40 20 40

At Line n1, an int [] object of three elements is created and ‘arr’ refers to this array object.

arr[0] = 10, arr[1] = 20 and arr[2] = 30;

Given expression at Line n2:

arr[i++] = arr[++i] = 40;

Multiple assignment operators are available, so lets group it first.

=> arr[i++] = (arr[++i] = 40); //Assignment operator is right to left associative

Above expression is valid, hence Line n2 compiles successfully.

Let’s solve the expression now. Left operand is ‘arr[i++]’ and right operand is ‘(arr[++i] = 40)’. Left operand is evaluated first.

=> arr[0] = (arr[++i] = 40); //i = 1

Right hand operand is evaluated next.

=> arr[0] = (arr[2] = 40); //i = 2

=> arr[0] = 40; //i = 2, arr[2] = 40.

Hence after Line n2, arr refers to int [] object {40, 20, 40}.

Given loop prints below on to the console:

40 20 40